

JPEG PROCESSING ENGINE FOR LOW PROFILE SYSTEMS

Kaixuan Mao

Yushan Huang

Donghui Wu

Anding Zhu

Lingxiang Zhou

FIELD OF INVENTION

[0001] This invention relates to Joint Photographic Expert Group (JPEG) compression technology for computer systems with limited resources.

DESCRIPTION OF RELATED ART

[0002] JPEG is designed to manipulate the characteristics of the human visual system. JPEG does this by discarding data conveying slight variances in color (e.g., chrominance) that are not easily recognizable to the human eyes to achieve grater compression of image data.

[0003] In JPEG, the source image is divided into a given number of blocks referred to as minimum coded units (MCUs). Each MCU consists several 8×8 blocks of pixel components from the source image. The height and width of the MCUs are determined by the largest horizontal and vertical sampling factors, respectively. The MCUs are processed from left to right and then top to bottom across the source image.

[0004] Fig. 1 is a flowchart of a method 10 for a conventional baseline JPEG engine executed by a system (e.g., a processor and a memory) to encode and decode a source image 40 in Fig. 2A. In steps 12 to 20, the system encodes source image 40. In steps 22 to 30, the system decodes the encoded bit stream.

[0005] In step 12, the system typically converts the RGB (red, green, and blue) values of the pixels in source image 40 to YCrCb (luminance and chrominance) values.

[0006] In step 14, the system separates the Y, Cr, and Cb components into three planes. Typically the system fully samples the Y values but downsamples the Cr and the Cb values as shown in Fig. 2A. The system then splits the Y, Cr, and Cb planes into 8×8 blocks. For a typical $\frac{1}{4}$ vertical and horizontal downsample of the Cr and the Cb values, MCUs are 16×16 blocks. The system interleaves the Y, Cb, and Cr 8×8 blocks to form the MCUs. In one example shown in Fig. 2B, source image 40 consists MCUs 1, 2, 3, and 4. MCU 1 consists of blocks Y1, Y2, Y3, Y4, Cr1, and Cb1, MCU 2 consists of blocks Y5, Y6, Y7, Y8, Cr2, and Cb2, and so forth.

[0007] In steps 16 to 20, the system encodes one MCU at a time. Within the MCU, the system encodes one 8×8 block at a time.

[0008] In step 16, the system performs forward discrete cosine transformation (FDCT) to the 8×8 blocks.

[0009] In step 18, the system performs quantization to the 8×8 blocks.

[0010] In step 20, the system performs entropy encoding (e.g., Huffman encoding) to the 8×8 blocks. After encoding all the MCUs that make up source image 40, the system has generated an encoded bit stream 50 where boundaries between the encoded MCUs are not known because the encoded MCUs do not have a fixed size and there are no markers demarcating their boundaries.

[0011] In steps 22 to 26, the system decodes one MCU at a time. Within the MCU, the system decodes one 8×8 block at a time.

[0012] In step 22, the system performs entropy decoding (e.g., Huffman decoding) to bit stream 50. By performing entropy decoding, the system is able to extricate the MCUs in the order which they were encoded in bit stream 50. However, the system cannot extricate one MCU before it entropy decodes one or more preceding MCUs in bit stream 50 because the encoded MCUs do not have a fixed size and there are no markers demarcating their boundaries. Thus, even though if only one MCU is requested to be decoded, all preceding MCUs in bit stream 50 must be entropy decoded in order to extricate the requested MCU. This requires the system to have sufficient CPU speed and memory to handle the entropy decoding of all the preceding MCUs.

[0013] In step 24, the system performs dequantization to the 8×8 pixel blocks.

[0014] In step 26, the system performs inverse discrete cosine transformation (IDCT) to the 8×8 pixel blocks.

[0015] In step 28, the system upsamples the Cr and the Cb values. In step 30, the system converts the YCrCb values to RGB values so source image 40 can be displayed.

[0016] As described above, method 10 needs to be implemented with a system having the sufficient CPU speed and memory to handle the entropy decoding of all the preceding MCUs of a requested MCU. Thus, method 10 is not well suited for systems for a low profile system such as an embedded system with a low CPU speed and a small memory. Thus, there is a need for a JPEG engine optimized for low profile systems.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] Fig. 1 is a flowchart of a method for a conventional baseline JPEG engine.

[0018] Figs. 2A and 2B illustrate a source image being encoded and decoded by the conventional baseline JPEG engine of Fig. 1.

[0019] Fig. 3 illustrates an encoded bit stream generated by the conventional baseline JPEG engine of Fig. 1.

[0020] Fig. 4 is a flowchart of a method for a JPEG engine to index and retrieve the MCUs in one embodiment of the invention.

[0021] Fig. 5 illustrates an encoded bit stream indexed by the JPEG engine of Fig. 4 in one embodiment of the invention.

[0022] Fig. 6 illustrates the use of the MCU index to crop a source image in one embodiment of the invention.

[0023] Fig. 7 illustrates the use of the MCU index to pan a source image in one embodiment of the invention.

[0024] Fig. 8 is a flowchart of a method for a JPEG engine that uses the MCU index to perform linear operations to the source image in the DCT domain in one embodiment of the invention.

[0025] Fig. 9 illustrates the use of the MCU index to transform the source image in one embodiment of the invention.

[0026] Fig. 10 illustrates the use of the MCU index to edit the source image from an edit action list in one embodiment of the invention.

[0027] Fig. 11 illustrates some exemplary user editing operations to a source image to generate a final image in one embodiment of the invention.

[0028] Figs. 12A, 12B, 12C, and 12D illustrate a data structure for tracking the editing operations and mapping the original MCUs in an original JPEG bit stream of a source image to the new MCUs in a new JPEG bit stream of a final image in one embodiment of the invention.

SUMMARY

[0029] In one embodiment of the invention, a method for indexing minimum coded units (MCUs) in a Joint Photographic Expert Group (JPEG) bit stream includes (1) entropy decoding a first MCU to determine a bit offset of a second MCU and a DC coefficient of the first MCU, wherein the first MCU immediately precedes the second MCU in the bit stream, and (2) indexing the second MCU by storing the bit offset of the second MCU and the DC coefficient of the first MCU in an index. The method may further include (3) receiving a request for the second MCU, (4) reading the index to determine the bit offset of the second MCU and the DC coefficient of the first MCU, and (5) entropy decoding the second MCU starting at its bit offset in the bit stream, wherein the entropy decoding the second MCU includes determining a DC coefficient of the second MCU using the DC coefficient of the first MCU.

DETAILED DESCRIPTION

[0030] Fig. 4 is a flowchart of a method 100 for a JPEG engine executed by a system (e.g., a processor and a memory) to incrementally index the minimum coded units (MCUs) in encoded bit stream 50 (Fig. 3) generated from source image 40 (Fig. 1) in one embodiment of the

invention. Bit stream 50 can be generated by any method that conforms to the baseline JPEG compression standard such as method 10 (Fig. 1).

[0031] In step 102, the system receives a request for an i^{th} MCU in bit stream 50. An MCU can be requested for many reasons. For example, a particular MCU may be requested to be retrieved for display so entropy decode, dequantization, and IDCT need to be performed. Alternatively, a particular MCU may be requested so it can be processed in the discrete cosine transformation (DCT) domain. Typically the i^{th} MCU is part of a series of MCUs that is requested for display.

[0032] In step 104, the system determines if the i^{th} MCU precedes the last indexed MCU in bit stream 50. The last indexed MCU is the MCU that was last to have its bit offset from the start of bit stream 50 (Fig. 5) recorded in an index file 130 (Fig. 5). In index file 130, the last indexed MCU is identified by a flag. If the i^{th} MCU precedes the last indexed MCU in bit stream 50, then step 104 is followed by step 116. Otherwise step 104 is followed by step 106. In one running example of method 100, assume that the i^{th} MCU is the fourth MCU (i.e., MCU 4) and the last indexed MCU is the third MCU (i.e., MCU 3). Thus, step 104 is followed by step 106.

[0033] In step 106, the system sets a variable “j” equal to the block number of the last indexed MCU. In the example, the system sets variable j equal to 3. Step 106 is followed by step 108.

[0034] In step 108, the system entropy decodes the last indexed MCU. By entropy decoding the last indexed MCU, the system determines the DC coefficient of the last indexed MCU and the start of the next MCU in the bit stream. The system then indexes the next MCU by storing the DC coefficient of the last indexed MCU and the bit offset of the next MCU in index file 130 (Fig. 5). The system stores the DC coefficient of the last indexed MCU so the DC coefficient of the next MCU can be decoded without decoding the last indexed MCU again. In the running example, the system entropy decodes MCU 3 and therefore determines the DC coefficient of MCU 3 and a bit offset C (Fig. 5) of the MCU 4 in bit stream 50. The system then indexes MCU 4 by storing the DC coefficient of MCU 3 and bit offset C of MCU 4 in index file 130 (Fig. 5). Step 108 is followed by step 110.

[0035] In step 110, the system updates the flag for the last indexed MCU to MCU j in index file 130 (Fig. 5). In the running example, the system resets the flag at MCU 3 and set the flag at MCU 4 in index file 130. Step 110 is followed by step 112.

[0036] In step 112, the system determines if the last indexed MCU precedes the requested i^{th} MCU in bit stream 50. To do so, the system determines if variable j is less than variable i . If so, then step 112 is followed by step 114. If variable j is not less than variable i , then step 112 is followed by step 116. In the running example, variable j is less than variable i (i.e., 3 is less than 4) so step 112 is followed by step 114.

[0037] In step 114, the system increments variable j by 1. In the running example, the system increments variable j by 1 so variable j becomes 4. Step 114 is followed by step 108. Steps 108, 110, 112, and 114 repeats until all the MCUs up to and including the i^{th} MCU have been entropy decoded and indexed, after which step 112 is followed by step 118.

[0038] In step 116, the system entropy decodes the requested i^{th} MCU. The system does this by looking into index file 130 (Fig. 5) for the bit offset of the requested i^{th} MCU and the DC coefficient of the preceding MCU. Using these values, the system entropy decodes the requested i^{th} MCU from the starting bit and restores the DC coefficient of the requested i^{th} MCU using the DC coefficient of the preceding MCU stored in index file 130. Step 116 is followed by step 118.

[0039] In step 118, the system determines if another MCU needs to be decoded. If so, step 118 is followed by step 104 and method 100 repeats until all the requested MCUs have been decoded and also indexed. Step 118 is followed by step 120, which ends method 100.

[0040] Instead of incrementally indexing the MCUs as they are requested as shown in method 100 of Fig. 4, the system can of course index the entire source image 40 at once in one embodiment of the invention. In this embodiment, the system would simply entropy decode each of the MCUs and record their bit offset and DC coefficient in index file 130.

[0041] MCU indexing can assist in several manipulation of source image 40. In a random JPEG cropping illustrated in Fig. 6, a child JPEG consisting of (x_0, y_0) to (x_1, y_1) MCU blocks is cut from an original mother JPEG. Thus, (x_0, y_0) to (x_1, y_1) MCU blocks need to be decoded and saved as a separate JPEG file.

[0042] Using conventional JPEG method 10, all the MCUs in the mother JPEG would need to be decoded in order to determine the boundaries between the MCUs of the child JPEG in the encoded bit stream. However, using JPEG method 100, only the MCUs in the child JPEG will need to be decoded if the first MCU in each row of the child JPEG has been indexed. For example, if (x_0, y_0) MCU has been indexed already, the system can move to the (x_0, y_0) MCU bit offset, restore the DC coefficients, and then decode the entire y_0 MCU row sequentially. The system can repeat the above steps for each row in the child JPEG if the first MCU in each row has been indexed. Thus, the system saves decoding time and buffer memory as the number of MCUs that need to be decoded is reduced.

[0043] In a JPEG panning illustrated in Fig. 7, a first region consisting of (x_0, y_0) to (x_1, y_1) MCUs is panned to a second region consisting of (x_0', y_0) to (x_1', y_1) MCUs. As can be seen, (x_0', y_0') to (x_1, y_1) MCUs have already been decoded. Thus, only (x_1, y_0) to (x_1', y_1) MCUs need to be decoded. Like random cropping, only the (x_1, y_0) to (x_1', y_1) MCUs will need to be decoded because the first MCU in each row of the second region has been indexed previously when the first region is decoded. Again, the system saves decoding time and buffer memory as the number of MCUs that need to be decoded has been reduced.

[0044] MCU indexing can also assist in the editing of a source image in the DCT domain. Fig. 8 is a flowchart of a method 200 for a JPEG engine executed by the system to perform linear pixel operations to the source image in the DCT domain in one embodiment of the invention. As DCT is a linear transformation, some linear pixel operations can be transferred to the DCT domain. If a linear pixel operation can be realized in DCT domain directly, IDCT and FDCT operation will be saved. When a linear pixel operation is needed for a few pixels within one MCU, the MCU indexing described above can save decoding time and buffer memory as the number of MCUs that need to be decoded is reduced.

[0045] In step 202, the system decodes the encoded bit stream to extricate the one or more requested MCUs. In one embodiment, method 100 described above is used to extricate the requested MCUs. The requested MCUs contain the pixels that will undergo the linear pixel operations. In one running example for method 200, the linear pixel operation includes scalar addition, scalar multiplication, or a combination thereof. Using method 100, the system will

only need to extricate the requested MCU that contains the pixels that will undergo the linear pixel operations without decoding the entire source image.

[0046] In step 204, the system performs dequantization to the requested MCUs.

[0047] In step 206, the system performs the linear pixel operation to the requested MCUs. In the running example, the system modifies the DCT blocks with the linear pixel operations.

[0048] In step 208, the system performs quantization to the requested MCUs.

[0049] In step 210, the system performs entropy encoding to the requested MCUs and then rewrites the MCUs in their proper order back into the encoded bit stream.

[0050] Method 200 of Fig. 8 can also be used to perform a pixel replacement operation, such as redeye removal, to one or more MCUs in one embodiment of the invention. In this embodiment, the system would perform the pixel replacement in step 206.

[0051] Fig. 9 illustrates the use of the MCU index to transform the source image in one embodiment of the invention. For example, source image 40 represented by encoded bit stream 50 is rotated 90 degrees clockwise to form an image 40' represented by an encoded bit stream 50'. Conventionally, all of MCUs 1 to 4 would need to be decoded, buffered, reordered, and then written to disk or another storage device. By using the MCU index, the system can determine where the boundaries between MCUs 1 to 4 and then entropy decode, rotate, and entropy encode one MCU at a time in the new order. Specifically, the system can now process MCU 3 before MCU 1, then process MCU 1, then MCU 4, and finally MCU 2 after MCU 4. Thus, the system does not need to modify the encoding procedure and the system will only need to buffer one MCU at a time for rotating source image 40.

[0052] Fig. 10 illustrates the use of a block editing list to edit the source image in one embodiment of the invention. In a low profile system, external storage input/output is very slow. If a MCU need several editing operations, it is very inefficient to save the MCU after each edit. Thus, the system creates a block editing list 230 for each MCU block to store the editing actions received from a user. The system can merge several editing actions and then encode each MCU one at a time to the final encoded bit stream. Referring to Fig. 10, assume the system needs to

enhance, rotate, and then crop MCU block 0. With the edit action list, the system can determine MCU 0 will be cropped out of the final image and therefore should not be processed.

[0053] Fig. 11 illustrates some exemplary user editing operations to a source image 250A to generate a final image 250D in one embodiment of the invention. Figs. 12A, 12B, 12C, and 12D illustrate a data structure for tracking the editing operations and mapping the original MCUs in an original JPEG bit stream of image 250A to the new MCUs in a new JPEG bit stream of image 250D in one embodiment of the invention. Specifically, Fig. 12A shows a data structure 270A of source image 250A prior to any editing operations. Data structure 270A stores MCU block numbers and their corresponding MCU block coordinates and MCU block editing lists, which were described above in reference to Fig. 10.

[0054] Referring to Fig. 11, the user first instructs the system to remove redeye from a region consisting of (x0, y0) to (x1, y1) MCUs in source image 250A to form an image 250B. In response to the user instruction, the system records the redeye action (“Redeye Removal”) in the editing action lists of (x0, y0) to (x1, y1) MCUs in data structure 270B shown in Fig. 12B.

[0055] Referring back to Fig. 11, the user next instructs the system to crop a region consisting of (r0, s0) to (r1, s1) MCUs in image 250B to form a child image 250C. In response to the user instruction, the system records the cropping action (“Not Available”) in the editing action lists of (r1+1, s0) to (r1+2, s1) MCUs that are cropped out of child image 250C in data structure 270C shown in Fig. 12C.

[0056] Referring back to Fig. 11, the user then instructs the system to rotate child image 250C to form final image 250D. In response to the user instruction, the system calculates the new block coordinates of all the MCU blocks in data structure 270D shown in Fig. 12D. The system also records the rotation action (“90CW”) in the editing action lists of (r0, s0) to (r1, s1) MCUs in data structure 270D. The rotation action is not recorded in (r1+1, s0) to (r1+2, s1) MCUs because their editing action lists indicate that these MCUs are cropped out of the final image 250D.

[0057] After all of the user editing actions, the system uses the final data structure 270D along with the MCU indexing method described above to encode the JPEG bit stream of final image

250D. Specifically, the system looks through the block coordinates stored in data structure 270D to determine the new order which the MCUs are to be encoded to represent final image 250D. In final image 250D, the first MCU is now (s0', r0') MCU. Using the entry of (s0', r0') MCU, the system reads the block number of (s0', r0') MCU in the original JPEG bit stream. Using the MCU indexing method described above, the system then retrieves (s0', r0') MCU from the original JPEG bit stream. The system next reads the block editing list of (s0', r0') MCU and processes the MCU block as instructed. The system then repeats this process for all the MCUs in the order which they appear in final image 250D.

[0058] Various other adaptations and combinations of features of the embodiments disclosed are within the scope of the invention. Numerous embodiments are encompassed by the following claims.